Vehicle Detection Project-5

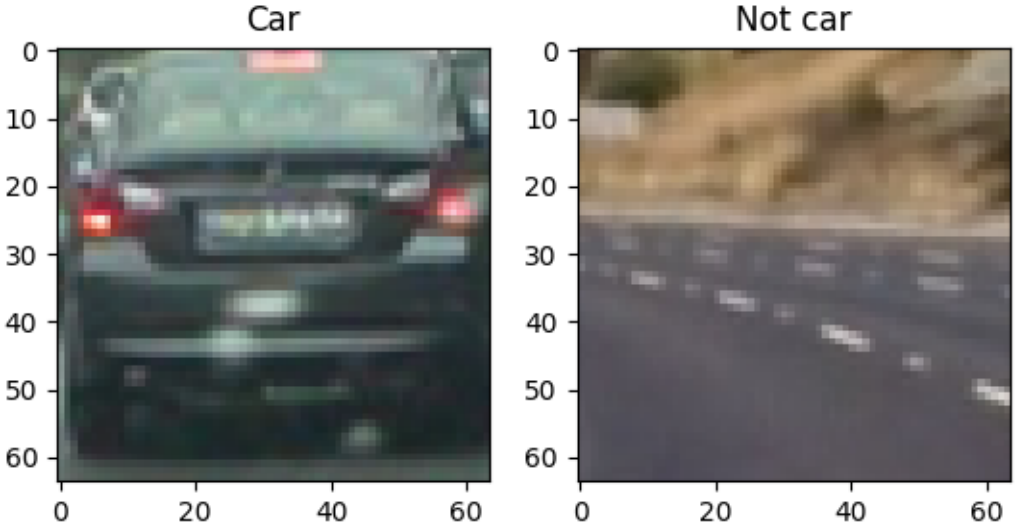
The following report for this project includes:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- A color transform and append binned color features, as well as histograms of color, to the HOG feature vector.
- Normalize these features and randomise and balance a selection for training and testing
- Train Support Vector Machine for the car and non car classification
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- Run the pipeline on a video stream and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the '**project_library/ExtractFeatures.py'** (lines 208-237)



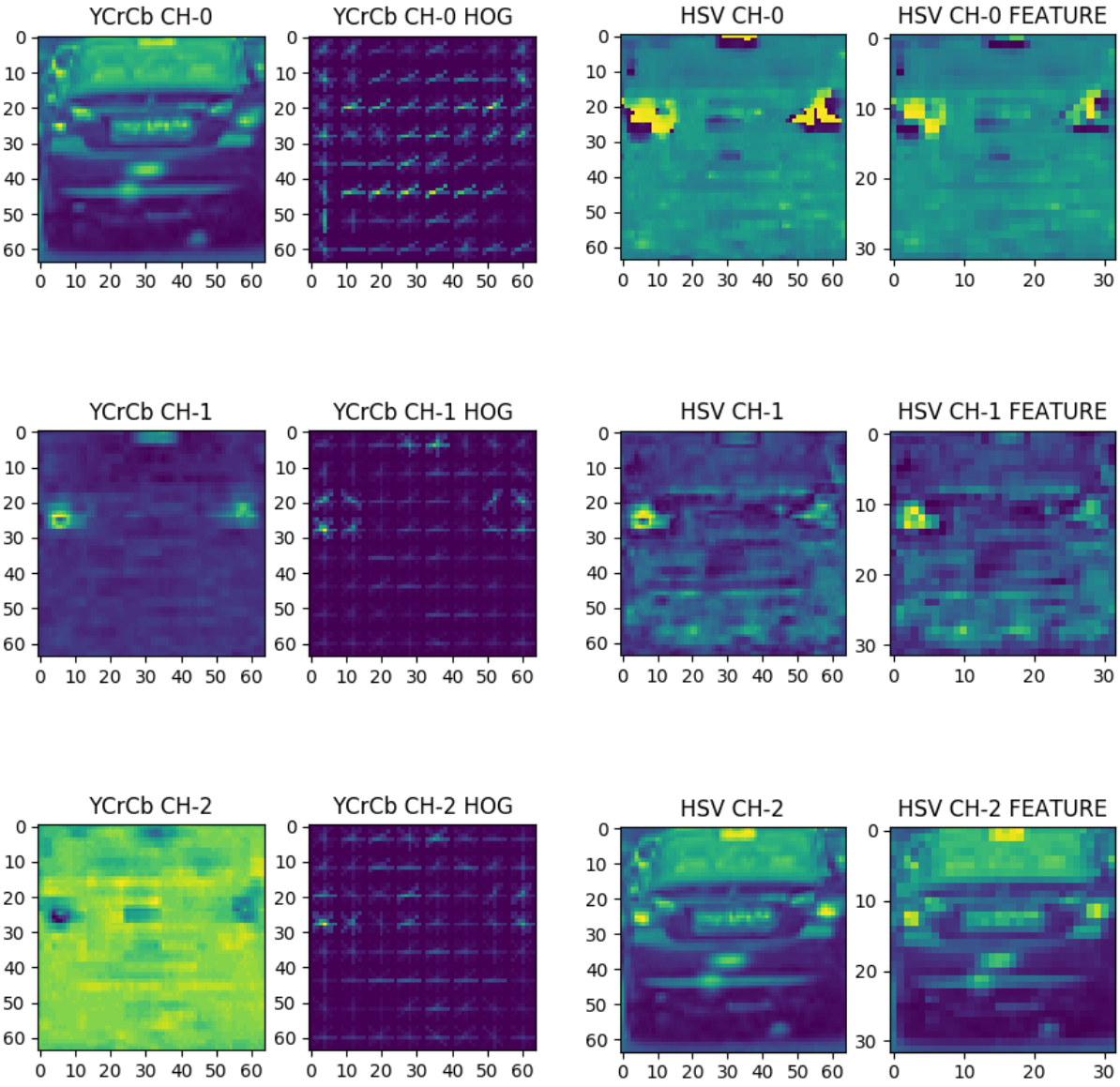
I started by reading in all the `vehicle` and `non-vehicle` images. Above is an example of one of each of the `vehicle` and `non-vehicle` classes.

Note: In order to maintain consistency with opencv in reading/manipulating .png and .jpg images. I am using 255 image scale through out. So I convert .png read in 0.0-1.0 scale to 255 scale. See **convertTo255() in 'project_library/ExtractFeatures.py'** in the line 29

I then explored different color spaces and different HOG parameters (‘orientations’, ‘pixels_per_cell’, and ‘cells_per_block’). I grabbed random images from each of the two classes and displayed them to get a feel for what the HOG output looks like.

Here is an example the HOG parameters of ‘orientations=8’, ‘pixels_per_cell=(8, 8)’ and ‘cells_per_block=(2, 2)’ using **YCrCb** as well as the **HSV** spatial color features (32x32).

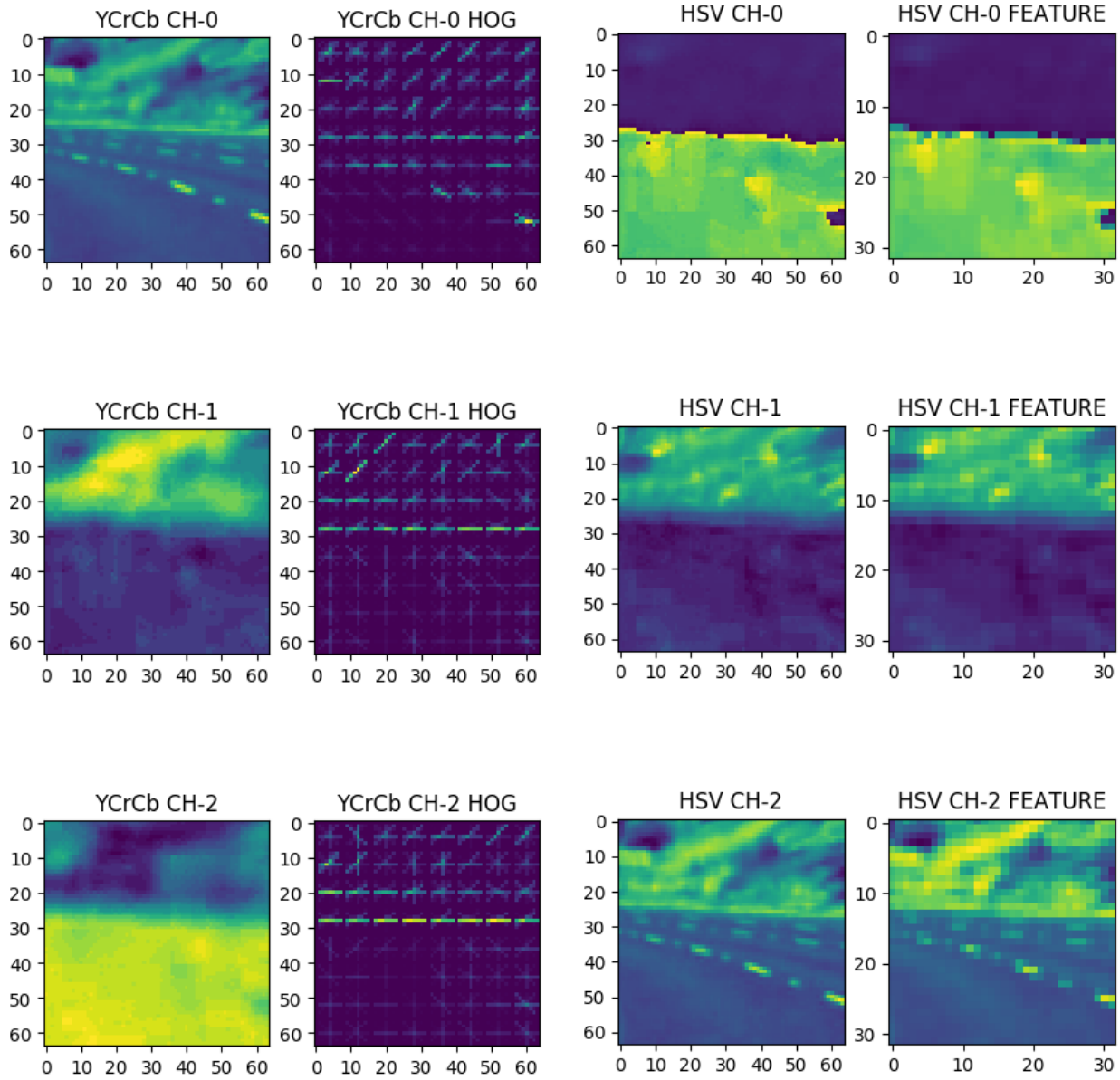
CAR (Hog: YCrCb, Spatial: HSV)



HOG 3 Channels: YCrCb Spatial 3 Channels: HSV

See test_bin_spatial, test_hog_feature (lines 169-204) in **‘project_library/ExtractFeatures.py’**

NOT CAR (Hog: YCrCb, Spatial: HSV)



HOG 3 Channels: YCrCb

Spatial 3 Channels: HSV

See test_bin_spatial, test_hog_feature (lines 169-204) in **'project_library/ExtractFeatures.py'**

2. Explain how you settled on your final choice of HOG parameters.

To get some idea on what the color spaces (e.g. YCrCb, LUV, HSV and etc) that I will use for this project. I tried the combination of these color spaces in the SVM training process to gauge what the color spaces and combination give the good performance (accuracy). The codes for this color channel search are in **project_library/GridSearchTrainClassifier.py**

For finding the hog parameters (also the spatial and histogram bin size), I am using this feature color space for extracting hog, spatial, and histogram features.
{'hist': 'YCrCb', 'spatial': 'RGB', 'hog': 'YCrCb'}

I tried various combinations of these parameters on the tested images.
(The detection is Hog sub-sampling with scales = [1.3, 1.5, 2.0])

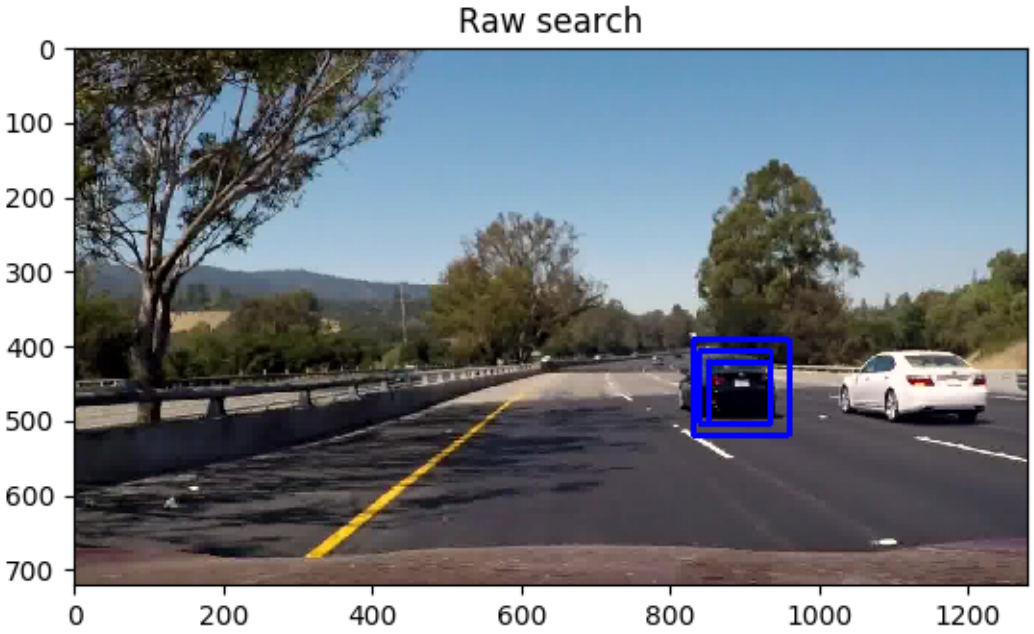
Finding the hog channel to use,
orientations = 9
pixels_per_cell = 8
cells_per_block= 2
hog_channel = 0, 1, 2 or 'ALL'
spatial_size = 16
hist_bins = 16

The results are discussed as follows:
Hog 0, 1, or 2 did not give a good result: lots of false detections

Raw detection: **Using hog channel 0**

Raw detection: **Using hog channel 1**

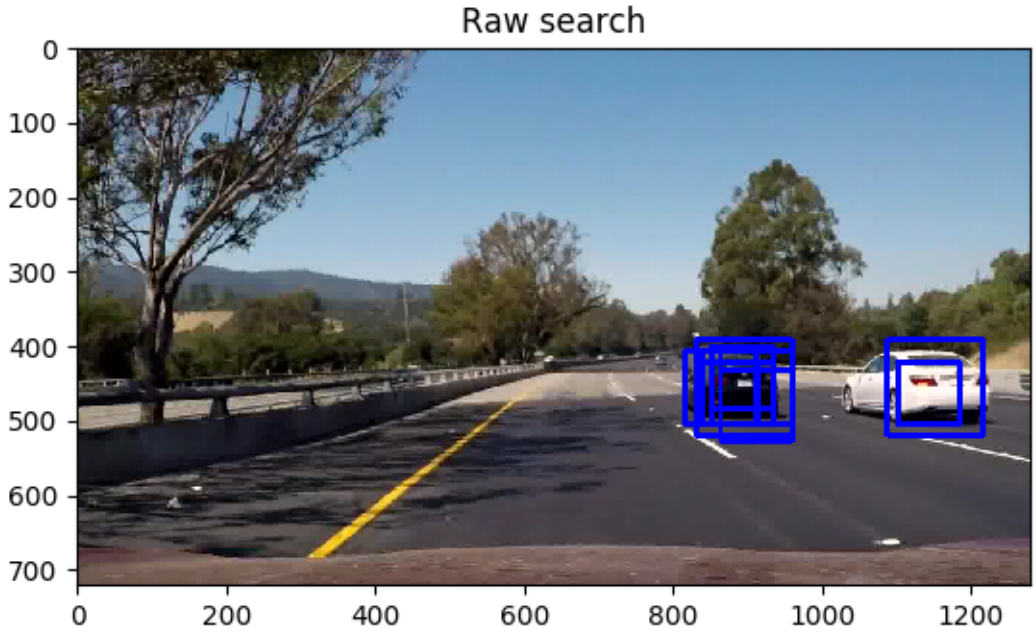
Raw detection: **Using hog channel 2**



Raw detection: **Using hog channel 'ALL'**

Hog 'ALL' is better but the white car still missing.

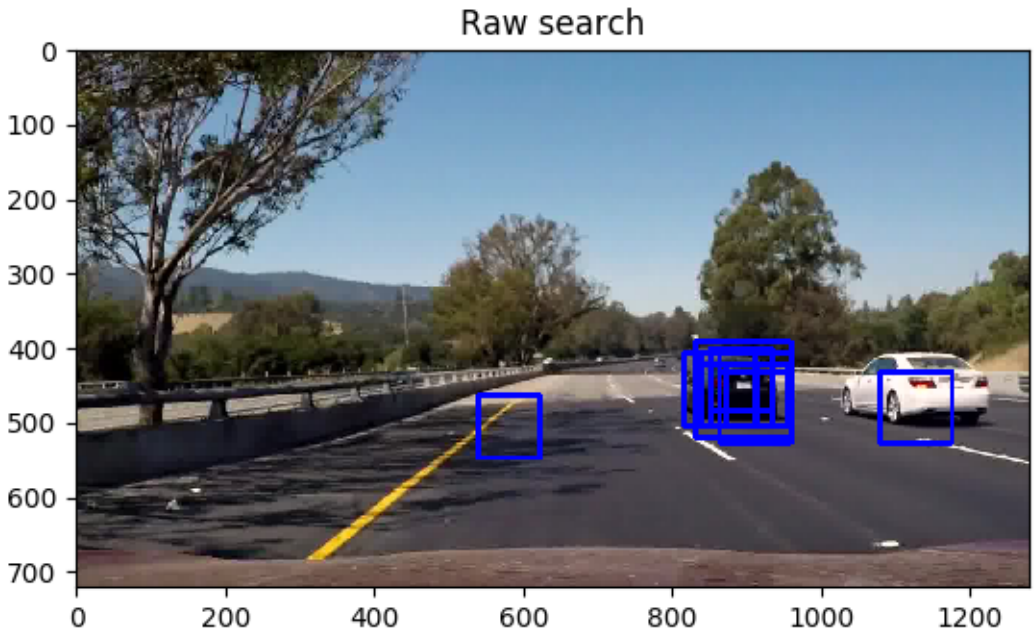
I then increased 'Spatial' and 'Histogram' bin size.



Hog 'ALL' + spatial 32x32 + histogram 32 (1.09 seconds)

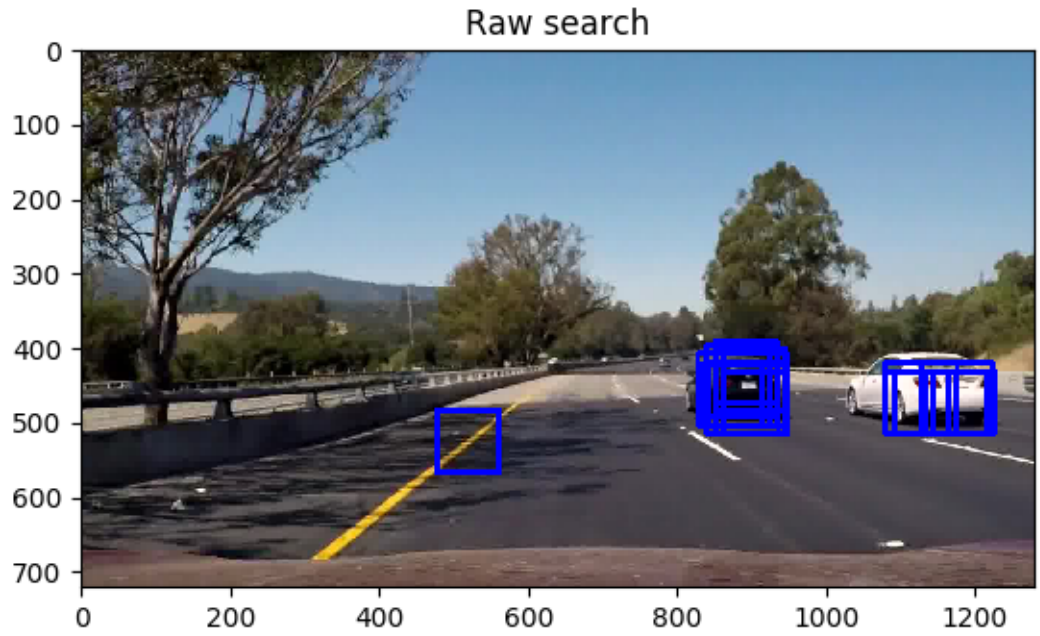
Hog 'ALL' + spatial 32 + histogram 32 is perfect (orientations (9), pixels_per_cell (8), cells_per_block(2))

I then tried different **orientations, pixels_per_cell** but they made it worse in term of performance (a false positive happened) and longer processing time. The results are shown belows.



Reducing orientation size to 8: a false detection happened

Hog ‘ALL’ + spatial 32 + histogram 32 is perfect with **orientations (8)**, pixels_per_cell (8), cells_per_block(2): processing time = 1.1 seconds



Reducing orientation size to 8 + pixels_per_cell to 4: false detection and processing time x4

Hog ‘ALL’ + spatial 32 + histogram 32 is perfect with **orientations (8)**, **pixels_per_cell (4),** cells_per_block(2): processing time = 4.31 seconds

With these experiments, I was settled with

Hog ‘ALL’ + spatial 32 + histogram 32 , orientations (9), pixels_per_cell (8), cells_per_block(2)

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

Read image names/Train and Test data split

I extracted the training/testing data from the image files. I balanced the cars and non cars images on both training and testing data set(GTI* files). In addition, **the time- series car images** are sorted in the time series order using the image tag names. I then split the first 80% for training and bottom 20% for testing. So the training data is different from the testing data to avoid over-fitting. The code for this is in **BuildImageNames()** function in **project_library/LoadData.py** (lines 26-28).

Data summary:
training car: 7032, training not car: 7174
testing car: 1760, testing not car: 1794
Number of training data: 14206
Number of testing data: 3554

Extract features

I then extracted the hog(*YCrCb*), spatial (*RGB*), histogram(*YCrCb*) features from each images for the SVM training in the **project_library/TrainClassifier.py** (lines 50 and 57). The code to extract the image features is in **project_library/ExtractFeatures.py** (lines 90-142)

SVM training

I trained a linear SVM using the default parameters on the training and testing features:

LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=1000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0)

The code for this is in **project_library/TrainClassifier.py** (lines 97).

Sliding Window Search

1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I decided to use the ***hog sub-sampling window algorithm*** that rescaling the image to different sizes in the search for the cars at various distances/sizes. The codes for the multiple scales search are implemented in the function **find_cars()** in **project_library/HogHeatMap.py** (lines 70- 170).

In the implementation, the input image is cropped out the unmarked region (lines 85). Finding any cars by iterating through the given scales, the cropped imaged is resized to the specific scale (lines 98-106). The hog, spatial, and histogram features are then extracted and transformed to match the trained image size of 64x64 pixels. For each scale iteration, the search window steps through the image to obtain the positive prediction and the location given by the bounding box (lines 135-168).

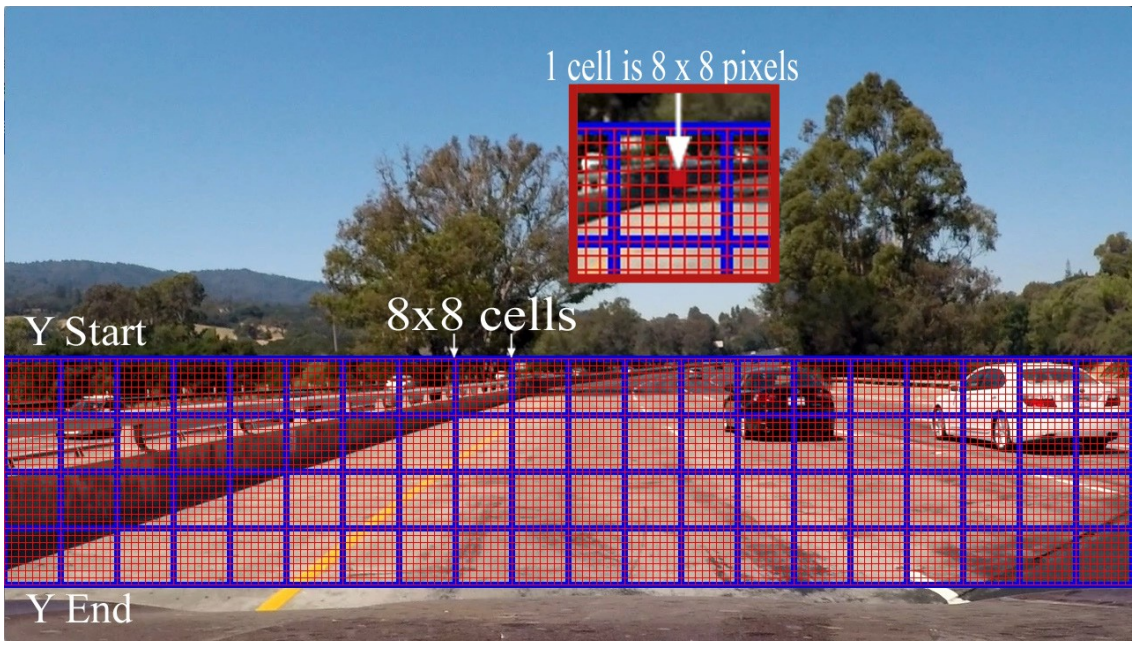


Image is from Udacity: Hog sub-sampling, window = 64, pix_per_cell = 8

The used settings are as follows:
Search area: ystart = 360, ystop = 640
Cells/step = 2 (100*6/8 = 75% overlap = 0.75*64 = 48 pixels or 16 pixels apart)
Scales = [1.3, 1.5, 2.0]

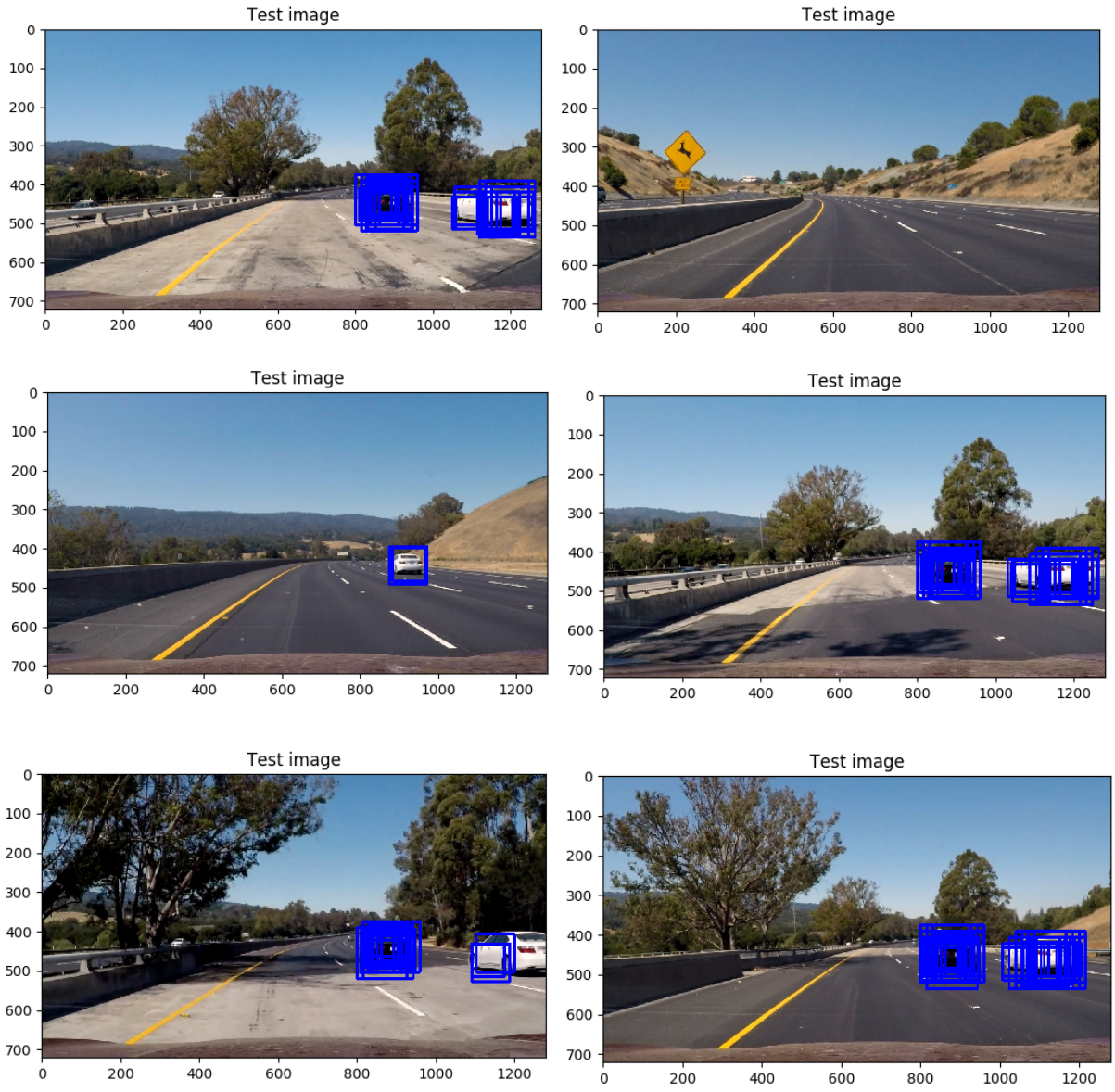
These scalings are estimated from the sizes of the vehicles that might appear on the images and we want to track them. To get some idea on the sizes and scaling by the trained image of 64x64 pixels
car size big: width = 200 pixels, height = 100 pixels → scale x = 3.1, scale y = 1.6
car size medium: width = 140 pixels, height = 65 pixels → scale x = 2.2, scale y = 1.0
car size small: width = 100 pixels, height = 50 pixels → scale x = 1.6, scale y = 0.8

The trained imaged size (cars) can be varied. So the scales was set according to these ranges. The above scales of 1.3, 1.5, 2.0 provided the good result in the detection for this implemented solution.

In addition, the window overlap was set to 75% to enhance a number of detections on the same vehicle. So we have a good portion of positive detections to build the more robust heat map and filtering.

2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on two scales using **3-channel of YCrCb HOG** features plus **spatially binned RGB color** and **histograms of YCrCb** in the feature vector, which provided a nice result. Here are some example images:



Video Implementation

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

See **project_video_out.mp4**

2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

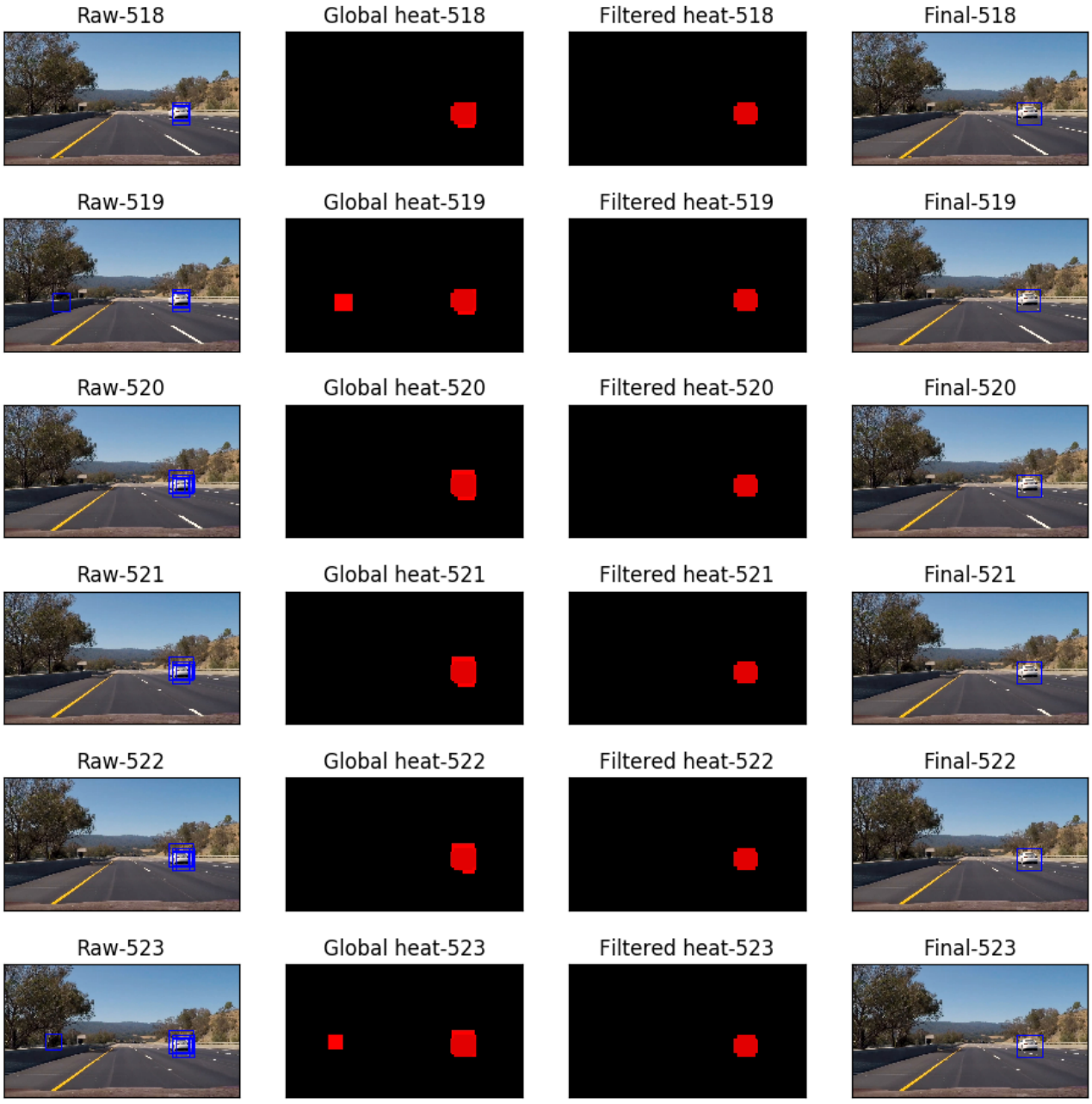
On filtering the false positive predictions, I used the similar approach as advised in the lesson. I slightly modified this technique to the global heat map update approach (lines 67 in **VehicleTracking_Video.py** and **update_globalmap()** function implemented in **HogHeatMap.py** lines 46) to keep track of the car positions overtime. In this approach, I recorded the positions of positive detections in each frame of the video by incrementing the pixel value by 1 and decreasing other area (negative detections) by 1 as well. So the heat value and position is building and

decreasing overtime. The limit of the heat values is controlled by the min and max threshold parameter. This limit will control the responsiveness in the vehicle detection and tracking (**VehicleTracking_Video.py** lines 33-36).

After the global map is updated on the current detection, it is then thresholded that map to filter out any false detection and to identify the vehicle positions (**VehicleTracking_Video.py** lines 33-36, **apply_threshold()** in **HogHeatMap.py** lines 24-48). I then used '*scipy.ndimage.measurements.label()*' to identify individual blobs in the heatmap. I then assumed each blob corresponded to the vehicle. I constructed bounding boxes to cover the area of each blob detected (**draw_labeled_bboxes()** in **HogHeatMap.py** lines 30-43).

Here's an example result showing the global and filtered heatmap from a series of frames of video, the result of 'scipy.ndimage.measurements.label()' and the bounding boxes then update/overlaid on every frame.

Here are six frames and their corresponding global and filtered heatmaps.



Raw detection **Global heat map** **Filter heat map** **Result output**

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

In this project, I used the support vector machine (SVM) classifier for detecting the vehicles in the image frame. The extracted features using Hog, Spatial color and Histogram for training the classifier are able to produce the good predictions. However, there are some places that the classifiers producing the false positive results due to changing conditions. Even though we can filter this out using the recorded heat map as implemented in this project or something similar. It still does not guarantee that these false detections would not slip through.

This implemented solution might fail in the places where the image contrast is low such as driving through overcasting shadow or the brightness is changed all the sudden. The potential solution to this problem is to train the classifier with these types of images.

For the future work, I believe a deep learning method such as Neural Network might produce a better result in recognising the cars from non-cars. In addition, we could use more than one classifier to weigh the prediction so we could filter out the low probability for the final output.